título

Procesamiento masivo de datos mediante Cassandra y Spark

Máster en Sistemas Informáticos Avanzados
Septiembre de 2016

Autor:

Xabier Zabala Barandiaran

Supervisores:

German Rigau i Claramunt
UPV/EHU
Iñigo Etxabe
Datik Información Inteligente S.L.

Agradecimientos

En primer lugar, quisiera expresar mi gratitud a las personas que han posibilitado la concepción y el desarrollo de la presente tesina. Agradezco a German Rigau i Claramunt, director del proyecto en la UPV/EHU, el asesoramiento ofrecido durante el transcurso del mismo. Doy también las gracias a Iñigo Etxabe, cofundador y CTO de Datik Información Inteligente, por facilitar los medios tecnológicos necesarios para llevar a cabo el trabajo y depositar total confianza en mi labor desde el primer día.

Agradecer, cómo no, a mis padres José Javier Zabala y María Pilar Barandiaran el esfuerzo desempeñado para allanar, en la medida que les ha sido posible, el camino que he recorrido hasta el día de hoy. Me congratula haber sabido responder satisfactoriamente a la confianza que han depositado en mí. Por lo dicho y por todo lo que suponen para mí, un beso enorme para los dos.

No me quisiera olvidar de aquellas personas que me han acompañado durante este maravilloso periplo. Doy las gracias a los amigos de siempre por el apoyo incondicional mostrado desde la distancia y a la gente que he tenido la fortuna de conocer durante los años de universidad. Quisiera agradecer especialmente a las personas que en este breve periodo se han ganado a pulso el privilegio a ser parte importante de lo que me resta de existencia, a los cuales me atrevo a mencionar aún con el temor de dejarme a alguna en el tintero: Adrián Núñez, Eider Irigoyen, Jaime Altuna y Marta García. No hallo palabras para expresar la gratitud que siento hacia vosotras.

Por último, pero no por ello menos importante, quisiera evocar a todos los docentes que han tomado parte en mi formación desde aquel Septiembre del 2008 y agradecer a todos ellos el conocimiento inculcado y el esfuerzo invertido en mi persona durante estos años.

Gracias de todo corazón a la gente mencionada en este breve capítulo por haber hecho de mí un mejor profesional y una mejor persona, además de darme las fuerzas necesarias para seguir evolucionando en ambos aspectos de cara al futuro.

Resumen

Proyecto Final del Máster en Sistemas Informáticos Avanzados. Estudio empírico sobre el rendimiento ofrecido por varias tecnologías emergentes en el campo del Big Data en comparación a una base de datos tradicional a la hora de operar en escenarios que requieren un almacenamiento y procesamiento eficaz de volúmenes masivos de datos.

Dos entornos de pruebas totalmente aislados han sido erigidos sobre la misma máquina física. En el primero, se ha construido un clúster compuesto por tres nodos virtuales que operan en la misma red privada. Dichos nodos han sido dotados de tecnología necesaria para el correcto funcionamiento de Apache Cassandra [9] y Apache Spark [20]. Sobre el segundo entorno, constituido por un nodo virtual de potencia equivalente al clúster ya mencionado, se ha instalado una instancia de MySQL Server. Una vez habiendo diseñado un conjunto de consultas equivalentes para ambas bases de datos, se ha procedido a poblar dichos sistemas de almacenamiento utilizando un data-set público de aproximadamente 25GB. Para finalizar, las consultas predefinidas han sido ejecutadas pudiendo así cuantificar el tiempo de respuesta ofrecido por cada entorno.

El estudio evidencia que a la hora de trabajar con volúmenes masivos de datos el binomio formado por Apache Cassandra y Apache Spark, además de ofrecer una solución totalmente escalable y tolerante a fallos, mejora sustancialmente el rendimiento ofrecido por MySQL Server. No obstante, para gozar de dichas ventajas, se antoja necesario invertir más tiempo en analizar exhaustivamente la naturaleza de los datos que se desean tratar.

Palabras Clave: Apache Cassandra, Apache Spark, Benchmark, Big Data, MySQL Server.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

Desde Aristóteles y su libro Segundos Analíticos [1] hasta Galileo, padre de la ciencia moderna, adalides del conocimiento han proclamado que un método de investigación basado en lo empírico y en la medición, sujeto a los principios específicos de las pruebas de razonamiento es el camino para conocer la verdad.

Hoy en día, época en la que los avances tecnológicos han posibilitado observar y medir de forma exhaustiva un gran abanico de fenómenos, la ingente cantidad de datos que se genera en el proceso es, a veces, intratable mediante las tecnologías convencionales, y por ende, es imposible extraer todo el conocimiento que atesoran. El problema, lejos de atenuarse, se acrecienta con el paso del tiempo. Estudios como el realizado por McKinsey Global Institute [2] estiman que el volumen de datos que se genera crece un 40 % cada año y auguran que entre 2009 y 2020 se verá multiplicado por 44 [12].

Para lidiar con dicha problemática, en los últimos años ha irrumpido la necesidad de desarrollar nuevas metodologías y tecnologías que permitan operar eficientemente sobre masas colosales de datos, dando como resultado el nacimiento del Big Data [10].

Empresas de renombre mundial, conscientes de los beneficios que el Big Data les puede reportar en diferentes facetas de su modelo de negocio, ya se han interesado en este fenómeno. De un estudio realizado entre los altos ejecutivos de las firmas que lideran el Wall Street se desprende que el 96 % tiene planeadas ciertas iniciativas relacionadas con el Big Data, y el 80 % ya tiene finalizada alguna [17].

[1] Órganon II de Aristóteles: Recopilación de obras Aristotélicas que incluye el libro Segundo Analíticos
[2] McKinsey Global Institute

1.1. Contexto

Datik Información Inteligente [3] es una empresa tecnológica perteneciente al Grupo Irizar [4] que desarrolla soluciones ITS destinadas a la gestión del trasporte, tanto ferroviario como por carretera y movilidad ciudadana.

Uno de los productos estrella de la entidad es el denominado iPanel, concentrador de información que ofrece al operador de transporte servicios de valor añadido en la gestión de la información generada por su flota. El funcionamiento del servicio se puede resumir mediante la Figura 1:

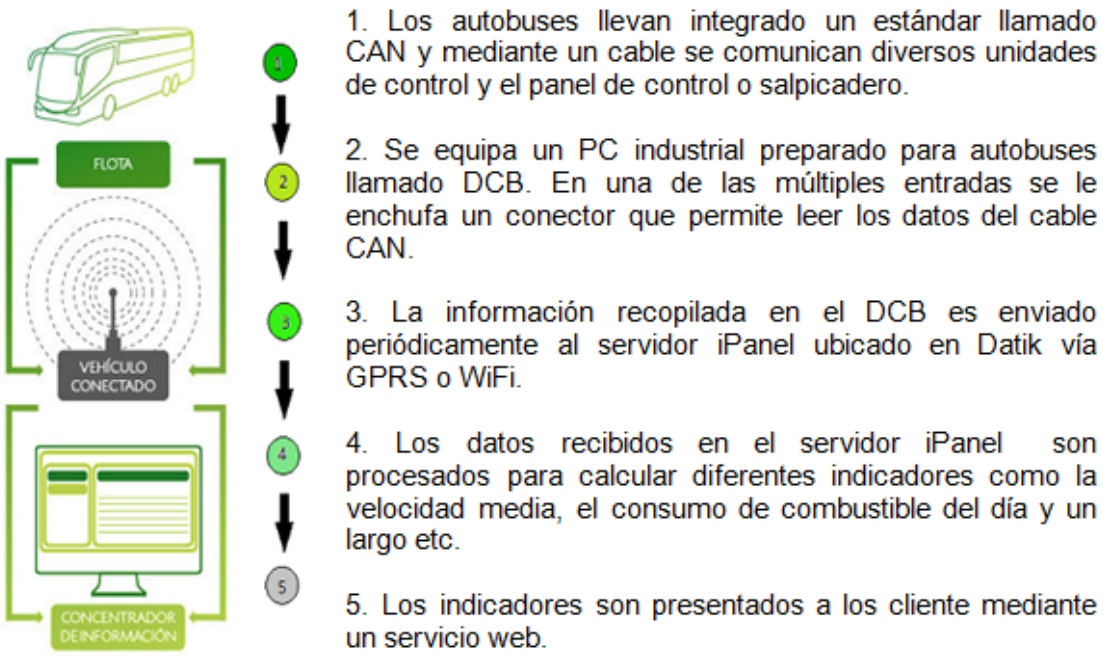

Figura 1: Funcionamiento resumido de iPanel

La incesante integración de nuevos vehículos a iPanel ha generado un crecimiento exponencial en el número de registros almacenados en ciertas tablas MySQL. Aunque el volumen actual no llega a suponer riesgo alguno para el funcionamiento del servicio, Datik tiene identificados varios escenarios en los que la situación se podría revertir, causando graves inconvenientes.

[3] http://www.datik.es/
[4] http://www.irizar.com/irizar/

Fiel reflejo de ello es el Cálculo de Indicadores: proceso de ejecución diaria que realiza operaciones aritméticas intensivas sobre datos almacenados en diversas tablas para después, agrupar los resultados en base a diferentes criterios. Siendo dichas tablas las que mayor crecimiento experimentan, el aumento del volumen de las mismas incrementa de forma desorbitada el tiempo necesario para finalizar el cálculo, pudiendo, en un futuro, llegar a tardar más de 24 horas y cancelar los indicadores que ofrecen la información del último día.

El objetivo del presente proyecto es proponer soluciones tecnológicas que solvente permanentemente el problema del proceso Cálculo de Indicadores y que a su vez, pueda valer para lidiar con otros obstáculos de la misma índole que pudieran emerger en un futuro.

1.2. Propuesta

Tal y como se ha mencionado en el apartado anterior, la problemática que envuelve al Cálculo de Indicadores es originado por el incremento exponencial de datos que dicho proceso ha de tratar. No sería descabellado pensar que la solución pudiese pasar por escalar verticalmente la máquina y afinar la configuración de MySQL. No obstante, ambas mejoras son limitadas, mientras que el volumen de los datos seguirá en aumento de forma inexorable, volviendo, tarde o temprano, a tener que lidiar con los mismos problemas del principio.

Siendo imposible reconducir la situación mediante las tecnologías tradicionales, en el presente proyecto se propone realizar un cambio de paradigma que implica migrar las tablas relacionadas con los Indicadores a un modelo distribuido que posibilite escalar la infraestructura horizontalmente. A su vez, se sugiere dividir el problema en dos apartados, almacenamiento y procesamiento, dotando la infraestructura de tecnología adecuada para ofrecer una respuesta eficaz a cada una de las partes.

En cuanto a almacenamiento se refiere, debido a la imperante necesidad de escarbar en hercúleos volúmenes de datos, se propone utilizar una base de datos no relacional. Dentro de la extensa y heterogénea gama de sistemas de almacenamiento NoSQL [5] existentes a día de hoy, se ha considerado idóneo el uso de Apache Cassandra [9], una base de datos distribuida del tipo clave-valor que ofrece alta disponibilidad sin un solo punto de fallo, un ratio de escrituras por segundo sustancialmente superior en comparación a sus homólogos [14] y velocidades de lectura nada desdeñables.

[5] `http://nosql-database.org/`

Por su parte, la tecnología seleccionada para el procesamiento ha sido Apache Spark [20]. Se trata de una plataforma de computación en clúster que ofrece una interfaz para programar operaciones paralelizadas y tolerantes a fallo que ha irrumpido con fuerza en el último lustro. A logrado desbancar tecnologías semejantes como MapReduce [3] gracias a la capacidad de almacenar en memoria los resultados intermedios del cómputo posibilitando así acelerar el procesamiento hasta 100 veces [18] en determinados escenarios.

Para cuantificar los beneficios aportados por las tecnologías propuestas se ha decidido utilizar un Dataset público de aproximadamente 25 GB que recoge la información de los trayectos realizados por los taxis amarillos de Nueva York durante el año 2015, ya que los datos actualmente almacenados por Datik se encuentran sujetos a una clausula de confidencialidad.[6]

1.3. Organización del documento

El presente documento abarca el estudio empírico realizado sobre el rendimiento obtenido mediante el uso conjunto de Apache Cassandra y Apache Spark en comparación al ofrecido por MySQL Server en escenarios que requieren un almacenamiento y procesamiento eficaz de volúmenes masivos de datos.

En este primer capítulo, se ha introducido la problemática que ha impulsado la creación del proyecto y expuesto la propuesta tecnológica para dar solución a la misma.

En el capítulo 2, se desgrana el funcionamiento de las tecnologías electas para resolver el problema, haciendo especial hincapié en las peculiaridades que les permiten ofrecer una mejora sustancial en comparación a las herramientas tradicionales.

Una vez en el capítulo 3, se detalla la praxis llevada a cabo para confeccionar los dos entornos de prueba. Por un lado, se describe la construcción de un entorno centralizado compuesto por un único nodo que alberga una instancia de MySQL y por el otro, la de uno distribuido, formado por tres nodos homogéneos sobre el cual operan Apache Cassandra y Apache Spark en consonancia.

El almacenamiento y la retribución de los datos constituyen el grueso del 4to capítulo. Primero, se expone la naturaleza de los datos que conforman el data-set y se enuncian las

[6] `http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml`

consultas que se desean realizar sobre este. Después, se presenta el diseño de las tablas utilizadas para almacenar dichos datos, incidiendo especialmente en la correspondiente a Apache Cassandra debido a las peculiaridades que presenta. Para finalizar, se detalla el preproceso realizado sobre los datos para almacenarlos de forma equivalente en ambas bases de datos.

En el capítulo 5, se publican los resultados obtenidos una vez habiendo ejecutado las consultas anteriormente definidas. Dichos resultados son representados mediante gráficas y analizados posteriormente para observar de forma nítida las diferencias existentes entre ambos entornos en lo que a tiempo de procesamiento se refiere.

Para finalizar, en el capítulo 6 se presentan las conclusiones y líneas futuras para el proyecto.

Fuera de la estructura general de la memoria, se puede encontrar la bibliografía citando todas las fuentes relevantes accedidas en el devenir del proyecto.

Capítulo 2

Análisis de las tecnologías propuestas

2.1. Apache Cassandra

Apache Cassandra es una base de datos distribuida que permite operar sobre grandes volúmenes de datos del tipo clave/valor. Concebida en el 2008 por los ingenieros de Facebook basándose en otros sistemas de almacenamientos distribuidos como Dynamo[4] y BigTable [2], se caracteriza por ofrecer una disponibilidad total y mayor escalabilidad lineal en comparación a otras bases de datos NoSQL[12]. Para ello, se basa en un conjunto de nodos homogéneos dentro de una red en anillo que se comunican mediante un protocolo P2P de replicación asíncrona, lo cual permite realizar operaciones de baja latencia para todos los clientes sin necesidad de un servidor maestro.

Hoy en día, es utilizada por infinidad de aplicaciones en negocios modernos, siendo la base de datos elegida por un tercio de las compañías que conforman la Fortune 100 [1]. Claro ejemplo de ello son empresas mundialmente conocidas como Apple, Facebook o NetFlix, los cuales utilizan Apache Cassandra como parte de su entramado tecnológico desde hace ya unos años. Cabe destacar que su uso no se limita al mundo empresarial. Muestra de ello es la acogida que ha tenido en el ámbito de la investigación, formando parte en experimentos punteros a nivel mundial como algunos de los realizados en el CERN [15].

[1] `http://fortune.com/2015/04/14/datastax-hp-sales-partnership/`

2.1.1. Funcionamiento

Las bases de datos NoSQL, en su mayoría, han sido concebidas para lidiar con un conjunto especifico de problemas. Debido a ello, para entender el funcionamiento de Cassandra cabe resaltar las peculiaridades que la distinguen de los demás sistemas de almacenamiento no relacionales.

Base de datos distribuida de alta disponibilidad

El Teorema de Brewer [6], también conocido como Teorema CAP , enuncia que un sistema de cómputo distribuido no puede garantizar simultáneamente las tres propiedades que se presentan a continuación, solo pudiendo cumplir dos de ellas al mismo tiempo, y acabar cumpliendo la restante tarde o temprano:

- **Consistencia**(Consistency): Todos los nodos ven la misma información al mismo tiempo.
- **Disponibilidad**(Availability): La garantía de que cada petición a un nodo reciba una confirmación de si ha sido o no resuelta satisfactoriamente.
- **Tolerancia al Particionado**(Partition Tolerance): El sistema sigue funcionando a pesar de que haya sido partido por un fallo de red.

Para una base de datos distribuida que promete la disponibilidad completa, el Teorema de Brewer implica la incapacidad de garantizar la consistencia total de los datos que almacena y la necesidad de esperar un tiempo indeterminado para que las réplicas de un registro modificado se actualicen correctamente.

Para lidiar con dicha restricción, Cassandra ofrece la posibilidad de afinar el nivel de consistencia [2] en cada consulta, sacrificando para ello parte de la disponibilidad. Así, las consultas de mayor disponibilidad corren el riesgo de ofrecer un dato obsoleto mientras que las de mayor consistencia pueden fallar si algún nodo caído provoca que no se pueda satisfacer el nivel de consistencia requerido.

Optimizado para escrituras

Una de las características a destacar de Cassandra es el elevado ratio de escrituras por segundo que es capaz de realizar, superando notoriamente a otras bases de datos NoSQL en este apartado [14].

[2] `https://docs.datastax.com/en/cassandra/2.1/cassandra/dml/dml_config_consistency_c.html`

Al ejecutar una consulta que desemboca en una escritura, los registros son almacenados en dos estructuras denominadas CommitLog[3] y Memtable. El primero, es un fichero donde se adjuntan los registros recibidos, mientras que el segundo, es una cache en memoria que apila dichos registros. Cuando ambas estructuras finalizan de almacenar los datos, Cassandra considera que la consulta ha sido ejecutado satisfactoriamente pudiendo pasar a procesar otras operaciones.

Cuando la Memtable[4] se llena empieza un proceso de vaciado en el cual, primero, se dilucida el nodo destinatario de cada registro. Después, dichos registros son escritos de forma secuencial en estructuras inmutables denominadas SSTable, ficheros que representan físicamente el contenido de una tabla. Si esta operación es interrumpida, se utiliza la información almacenada en el CommitLog para recuperar las escrituras perdidas. Finalmente, cuando todos los registros han sido transferidos a su correspondiente SSTable, el CommitLog es purgado.

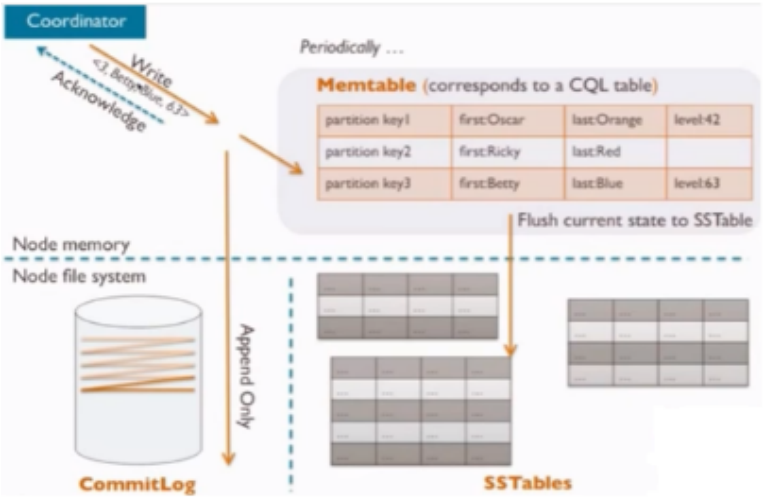

Figura 2: Almacenamiento de datos en Cassandra

Debido a la naturaleza inmutable de las SSTable, cada vez que la Memtable se vacía un nuevo fichero es generado para representar una tabla ya existente. Ello puede acarrear problemas de eficiencia a la hora de consultar la información de dicha tabla, ya que será totalmente necesario leer todos los ficheros correspondientes. Para sobreponerse a este inconveniente, Cassandra posee un mecanismo llamado compactación que, entre otras acciones, unifica las SSTable que representan una misma tabla eliminando los ficheros

[3] `https://wiki.apache.org/cassandra/Durability`
[4] `https://wiki.apache.org/cassandra/MemtableSSTable`

restantes.

Replicación asíncrona sin maestro

Las bases de datos distribuidas ofrecen la posibilidad de replicar los datos que se almacenan en ellas, pero dicha opción implica la necesidad de conocer continuamente, por parte de todos nodo del sistema, el estado de cada réplica. Por lo general, el mencionado inconveniente se resuelve mediante un modelo maestro-esclavo.

En Cassandra, no obstante, gracias a un conjunto de mecanismos que se presentan a continuación, cada nodo es capaz de obtener dicha información de forma local sin depender de un nodo maestro, obteniendo un clúster totalmente homogéneo.

Gossip Protocol[5]: Protocolo de comunicación peer-to-peer que intercambia el estado del propio nodo y de aquellos que conoce de forma periódica.

Cada mensaje Gossip tiene una versión asociada a él, gracias al cual el nodo que recibe dicho mensaje puede comparar con la información que tiene de antemano y actualizar su conocimiento en caso de ser necesario.

En el caso concreto de Cassandra, el intercambio se realiza cada segundo entre los nodos vecinos, logrando así que, al propagarse la información por el anillo, todos los nodos del clúster aprendan sobre el resto de manera rápida.

Failure Detector[1]: Conjunto de métodos que usando mensajes Gossip sugieren de forma local si otro nodo del sistema esta caído o en estado transitorio.

Dentro de la extensa gama de métodos existentes, Cassandra utiliza el Accrual Failure Detector[8].

Además ofrece la posibilidad de configurar un umbral por cada nodo que tenga en cuenta el rendimiento de red, carga de trabajo, u otras variables. Al cotejar el valor ofrecido por el detector con dicho umbral, Cassandra es capaz de dilucidar si un nodo esta caído, y en caso afirmativo, redireccionar las peticiones a otro activo.

Merkle Tree[11]: Es un árbol hash donde los nodos padres más altos son, a su vez, hashes de sus respectivos hijos. La ventaja principal es que cada rama del árbol puede ser comprobada de forma independiente, sin la necesidad de descargar todo el conjunto de datos.

La implementación de Anti-Entropia[7] en Cassandra, proceso encargado de comparar todas las réplicas de cada dato que existe en el clúster y actualizarlos a la versión más reciente, genera un Merkle Tree por cada Column Family durante el proceso de compactación, los cuales se almacenan solo hasta enviarlos a los nodos vecinos.

La praxis descrita implica enviar un exceso de datos por la red pero ahorra en operaciones I/O del disco local, lo cual es preferible para conjuntos de datos muy grandes.

Sin punto único de fallo

La ejecución de una consulta es considerada satisfactoria si el clúster cumple con los requisitos de consistencia especificados en la misma. Esta simple premisa se embarulla en un entorno real en donde los nodos de la infraestructura pueden cambiar de estado en cualquier instante y además, seguir cumpliendo con los requisitos de consistencia, dando como resultado el no poder actualizar las réplicas que residen en los nodos caídos.

Cassandra implementa varios mecanismos con el objetivo de evitar la cancelación de las consultas en dicho escenario:

Rapid Read Protection[5]: El nodo del clúster que recibe una consulta es denominado coordinador y cumple con el cometido de identificar la máquina menos ocupada entre los que contienen una réplica del registro requerido y redirigir la consulta hacia ella.

Si en ese preciso instante el nodo electo deja de estar disponible, la técnica Rapid Read Protection permite al coordinador escoger otro nodo entre los candidatos y redirigir la petición hacia este último, evitando así tener que devolver un mensaje de error al cliente.

Hinted Handoff[6]: Al persistir una escritura, técnica que almacena en una tabla interna del nodo coordinador los registros que no han podido ser replicados a otras máquinas. Cuando recibe un mensaje Gossip indicando que dichas máquinas vuelven a estar activas, se encarga del envío de los registros correspondientes.

2.1.2. Cassandra Query Lenguage (CQL)

El código encargado de tratar las consultas que llegan al clúster está generado mediante Apache Thrift[16], una interfaz para implementar llamadas a procedimientos remotos[13] que ofrece la posibilidad de conectarse a Cassandra utilizando diversos lenguajes

[5] `http://www.datastax.com/dev/blog/rapid-read-protection-in-cassandra-2-0-2`
[6] `http://www.datastax.com/dev/blog/modern-hinted-handoff`

de programación.

No obstante, la gran curva de aprendizaje que requería amoldarse a la sintaxis generada por Thrift actuaba de muro para un profesional acostumbrado a trabajar con SQL. Para evitar este inconveniente, los desarrolladores han optado por su propio lenguaje de consultas, el denominado Cassandra Query Lenguage (CQL).

Aún siendo sintácticamente tan parecido a SQL, presenta ciertas restricciones debido a que es un lenguaje de consultas de una base de datos distribuida. Por ejemplo, no ofrece la posibilidad de realizar operaciones como JOIN y es totalmente necesario especificar todas los atributos que componen la clave primaria a la hora de realizar cualquier consulta de filtrado o de actualización en la tabla. La única operación que no cumple esta restricción es un select que contenga la clausula where, ya que, al definir la clave primaria Cassandra indexa de forma automática todos sus componente, posibilitando más tarde hacer uso de ellos en este caso concreto.

2.2. Apache Spark

Apache Spark[20] es un proyecto open source de computación en clúster. Desde el principio fue diseñado para poder ejecutar algoritmos iterativos en memoria sin la necesidad de almacenar en disco los resultados intermedios generados durante el proceso. Esta peculiaridad permite que los procesamientos llevados a cabo con Spark puedan llegar a ser, en algunos casos concretos, 100 veces más rápidos que los de MapReduce[3].

A mediados de 2014, coincidiendo con el lanzamiento de la primera versión, alcanzó la cifra de 465 colaboradores, convirtiéndolo en el proyecto más activo entre los relacionados con el Big Data dentro de la Apache Software Fundation.

La base del proyecto es el denominado Spark Core. Proporciona envío distribuido de tareas, planificación y funciones básicas de entrada salida. La abstracción fundamental de programación se llama Resilient Distributed Datasets[19], una colección lógica de datos particionados a través de las máquinas que se expone mediante una API integrada en lenguajes como Java, Python y Scala.

2.2.1. Funcionamiento

Para el funcionamiento de Spark, es condición sine qua non que los nodos de la infraestructura tengan acceso a la totalidad de los datos que se desea tratar. Ello implica que

para procesar un fichero de 25GB, cada nodo tendría que poseer una copia del mismo almacenado en su disco. Esta praxis es inviable, ya que más allá de los problemas de consistencia que generaría, para nada es eficiente ocupar la memoria de todos los nodos con información redundante y procesar el fichero entero cuando en realidad se va a hacer uso de una pequeña porción de dichos datos en cada ejecución.

Las bases de datos distribuidas como Cassandra solventan los problemas anteriormente mencionados. Se encargan de distribuir los datos entre diferentes nodos del clúster, ofrecen la posibilidad de acceder a ellos desde cualquier punto y mantienen la consistencia de los mismos a cambio de sufrir una pequeña latencia en el caso de requerir información almacenada en otro nodo de la infraestructura.

Al ejecutar una aplicación que opera con Spark, un componente denominado driver es lanzado. Debido a la necesidad de obtener recursos (CPU y memoria) para llevar a cabo la computación que se le ha encomendado, se comunica con un nodo del clúster que, mediante especificación previa, adopta el rol de maestro. Éste pregunta a todos los nodos que conforman la infraestructura sobre la cantidad de recursos disponibles que poseen y así asignarles los executors correspondiente. Las máquinas que alojen al menos un executor pasan a denominarse worker y a partir de este momento, podrán comunicarse directamente con el Driver para poder recibir las tareas que éste le envíe.

Un executor es una unidad de trabajo que se encarga de computar las tareas que le encomienda el driver. El número de executors que puede albergar cada worker está directamente relacionado con el número de procesadores que este posee. De la misma forma, es posible repartir la memoria RAM que dispone el nodo worker entre varios executors. Spark permite modificar ambos parámetros programáticamente permitiendo así poder amoldarse a las particularidades de cada ejecución.

Para transferir el código del programa, residente en la máquina del driver, éste adopta el rol de servidor e intenta enviar dicho código a los workers. Si el fichero JAR que contiene el código ha sido recibido correctamente por sus destinatarios, estos responden mediante un ACK y en caso contrario, se vuelve a intentar el envío un número determinado de veces. Una vez llegado al máximo de reintentos, el worker que no haya enviado el ACK es considerado como caído, quedando los executors que albergaba fuera del posterior reparto de tareas.

A la hora de realizar operaciones en Spark, el objeto estrella es el denominado Resilient Distributed Datasets (RDD)[11]. Se trata de una abstracción que mediante diferentes APIs disponibles para Java, Scala y Python permite manipular datos distribuidos por los diferentes nodos del clúster como si estuvieran almacenados de forma local. Este

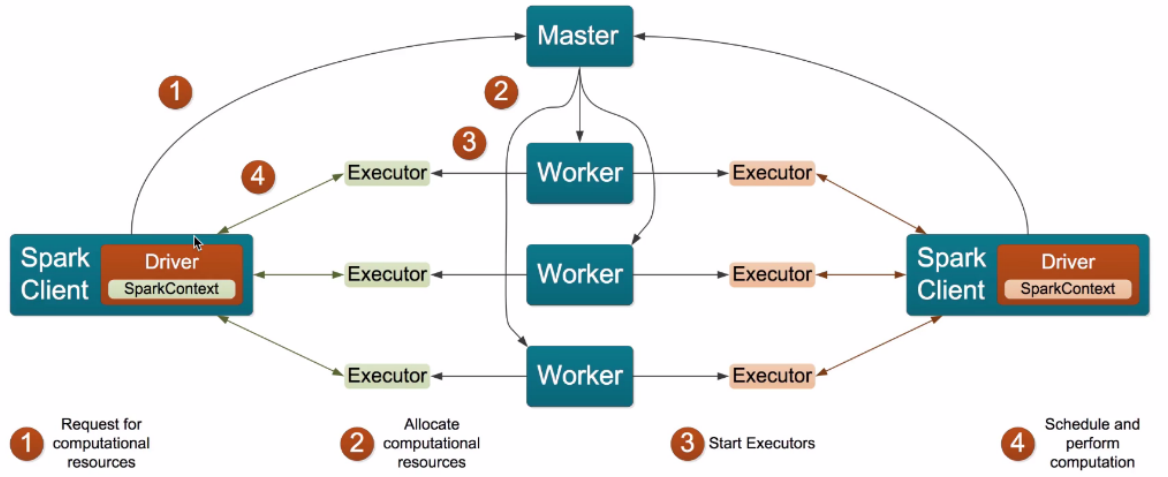

Figura 3: Arquitectura Spark

objeto es inmutable, lo cual implica que una vez creado no se le pueden añadir nuevos elementos o eliminar los existentes, solo aplicar transformaciones y acciones sobre el.

Las operaciones que se pueden realizar sobre las RDD se agrupan, tal y como se ha adelantado antes, por transformaciones y acciones. Las primeras transforman un RDD en otro según el criterio indicado y las segundas realizan modificaciones sobre los datos almacenados en dichas RDD. Cabe destacar que las transformaciones en Spark son operaciones "lazy", lo cual implica que en realidad cada nodo memoriza la secuencia de transformaciones que ha de realizar y los procesa cuando una acción es ejecutada.

Una vez terminada la primera fase en la que los executor son creados y enlazados con el driver, éste último empieza a analizar la estructura del código y genera un grafo DAG (Directed Acyclic Graph) con las operaciones que se realizan sobre la RDD. Partiendo de ese grafo genera un job por cada operación de tipo acción que encuentra y dentro de cada job separa la ejecución en diferentes stages según las dependencias que existan entre operaciones. Por último, cada stage es dividido por defecto en unidades de 64MB y a cada unidad resultante se denomina task, el cual es enviado a un executor para ser procesado. El tamaño de cada task puede ser modificado programáticamente, pudiendo de esa forma manipular el número de task que un executor deba ejecutar.

El driver, una vez habiendo recibido los resultados de todas los task que ha repartido, enviará un mensaje a los executors indicando que el procesamiento ha sido finalizado y calculará el resultado final ofreciendo la posibilidad de, por ejemplo, almacenarlo en una base de datos distribuida como Cassandra.

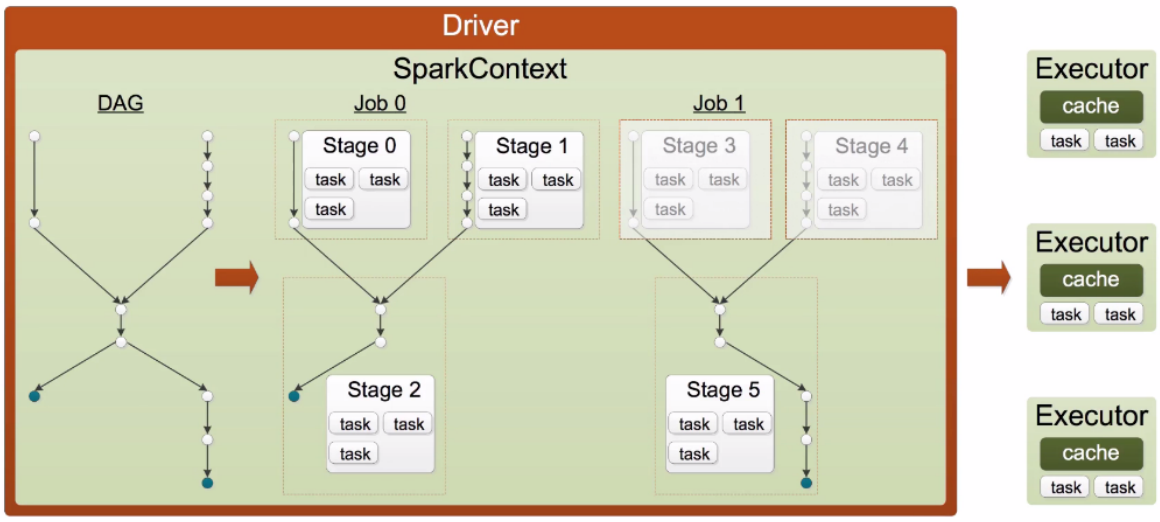

Figura 4: Arquitectura Spark

Capítulo 3

Entorno de pruebas: Construcción

En las próximas líneas se detallan las directrices seguidas a la hora de confeccionar los entornos de prueba posteriormente utilizados para cuantificar el rendimientos ofrecido por las tecnologías analizadas.

Para ello, se empezará por especificar las características físicas y la confección de los dispositivos utilizados para erigir dichos entornos:

3.1. Características de las máquinas físicas

Ordenador portátil: Terminal encargado de lanzar diversas peticiones contra el servidor y recolectar los resultados computados por este cuantificando el tiempo transcurrido entre ambos eventos.

- Procesador: Intel(R) Core(TM) i-5 4200M CPU @ 2.50 GHz
- Memoria RAM: 8 GB
- Disco Duro: Western Digital WD5000LPCX-24C6HTO SSD 500GB
- Sistema Operativo: Windows 10
- Adaptador Red: Qualcomm Atheros AR8172/8176/8178 PCI-E Fast Ethernet Controller (NDIS 6.30)

Ordenador de sobremesa: Terminal que adquiere el rol de servidor dentro del entramado. Se encarga del tratamiento de los datos mediante el uso de las tecnología que se analizan en el presente proyecto.

- Procesador: AMD FX(tm)-8350 Eight-Core Processor 4.00GHz
- Memoria RAM: 32 GB
- Disco Duro: Western Digital WD5000AAKX-22ERMA0 SSD 500GB
- Sistema Operativo: Windows 10
- Adaptador Red: Killer e2200 Gigabit Ethernet Controller (NDIS 6.30)

Ambas máquinas se encuentran enlazadas a un *switch* tp-link-tl-sf1008d mediante sendos conectores Fast Ethernet, permitiendo intercambiar datos a una velocidad de 100 Mb/s.
El switch, a su vez, está conectado a un *router* que le proporciona acceso al exterior de la red privada, el cual sólo será utilizado para descargar contenido relativo al preparativo de las pruebas.

3.2. Confección de las máquinas virtuales

Las máquinas virtuales juegan un papel vital en el desarrollo de las pruebas. Gracias a ellas, es posible aislar completamente ambos entornos dentro del ordenador de sobremesa y además, permiten la creación de un clúster sin necesidad de adquirir más dispositivos.

Con el objetivo de no realizar las mismas configuraciones básicas reiteradamente, se ha decidido crear una máquina virtual básica y una vez configurada, clonarla tantas veces como instancias hagan falta para llevar a cabo las pruebas. Al considerar esta configuración un procedimiento trivial y carente de especial interés en el desarrollo del presente proyecto, los entresijos han sido omitidos, describiendo solamente la motivación de cada paso.

1. *Crear la máquina virtual básica*:

   Instancia creada mediante VirtualBox utilizando una imagen de Ubuntu Server 16.04 LTS. Se le ha asignado una memoria física de 75 GB y mantenido tanto el

número de procesadores como la memoria RAM que se le asigna por defecto, ya que estas dos últimas características son modificables posteriormente.

2. *Asignar una IP estática*:

   Todos los clones realizados sobre la instancia básica operarán como servidores, por lo que no es de recibo que su IP pueda variar cada vez que el servicio de red o la máquina misma se reinicien. Para evitar cualquier inconveniente de este estilo, es totalmente necesario configurar una interfaz de red con una IP estática por la cual se podrá acceder a la máquina.

3. *Cambiar el adaptador de red a modo puente*:

   Al crear una máquina virtual con VirtualBox, el adaptador de red predefinido es el NAT. Ello implica que dicha máquina se encontrará aislada en una red lógica y solo podrá acceder a otros dispositivos pasando por la máquina física y haciendo uso de su IP.
   Modificando el adaptador de red al modo puente, se consigue extender la red privada hasta el nodo virtual eliminando así toda dependencia con la máquina física en cuanto a comunicaciones se refiere, posibilitando que acceda al resto de las máquinas que residen en la misma red privada.

4. *Instalar Java 8*:

   El software que se va a arrancar en las instancias clonadas se ejecuta sobre la Máquina Virtual de Java(JVM), por lo que es necesario tener una versión de Java instalada en la máquina virtual base.

```
1  sudo add-apt-repository ppa:webupd8team/java
2  sudo apt-get update
3  sudo apt-get install oracle-java8-installer
```

5. *Instalar SSH Server*:

   Aunque no se considere estrictamente necesario llevar a cabo este último paso, es de gran ayuda poder acceder a las máquinas virtuales mediante SSH y transferir

archivos utilizando SFTP.

```
1   sudo apt install openssh-server
```

Una vez terminado de confeccionar la máquina virtual básica, solo falta arrancar y medir ciertas metricas para asegurarnos que es posible conectarse al portatil en un tiempo reducidos.
// Comprobar la latencia y el numero de hoops
// La maquina virtual prototipo ha sido finalizado, guardar una copia por si se desea añadir mas nodos en un futuro, a cada uno habra que cambiarle la memoria ram y procesadores

3.3. Entorno centralizado

emular el entorno que actualmente datik posee y el rendimiento que da

1. Nested item 1
2. Nested item 2

3.4. Entorno distribuido

1. Nested item 1
2. Nested item 2

A continuación se detallan los pasos más relevantes llevados a cabo para construir y configurar el entorno distribuido o clusterizado. Instalar las máquinas virtuales que operan cual nodos y dotarlos de un sistema operativo se ha considerado un procedimiento trivial y carente de interés por lo cual estos pasos han sido omitidos.

///Configuración de los fichero etc/hosts
Para finalizar, es indispensable que cada máquinas conozca a las otras que componen el cluster para que estos actúen de forma cooperativa. Dicha información les será transmitida mediante el fichero /etc/hosts.
En el siguiente recuadro se puede apreciar la apariencia que han de tener los ficheros /etc/hosts de todas las máquinas del cluster:
192.168.1.201 master 192.168.1.202 slave1 192.168.1.203 slave2

El motivo por el cual unos pasos atrás se ha asignado un hostname a cada máquina virtual sido para, mediante este fichero, relacionar cada host con su respectiva IP. A priori, no parece que sea estrictamente necesario realizar esta configuración, ya que una dirección IP es elemento suficiente para identificar un terminal en la red, pero a la larga ofrece diversas ventajas. Por un lado ventajas operativas: un nombre es más fácil de relacionar a un nodo gracias a la semántica inherente a el, algo que no pasa con una IP, que a la postre, cambia si el nodo es trasladado de una red a otra. Por otro lado, las ventajas funcionales como las que se van a poder observar en el siguiente apartado; por ejemplo, que algunas variables de Apache Cassandra, en caso de no ser configurados, recurren a este fichero para conocer la IP de una máquina.

3.4.1. Configuración de Apache Cassandra

Cassandra puede ser instalada mediante paquetes deb o rpm pero en este proyecto no se ha optado por ninguna de esa vía. Al haber elegido la instalación manual, será necesario crear los siguientes directorios en cada uno de los nodos para que el archivo de configuración cassandra.yaml pueda guardar y acceder a la información generada durante la ejecución:

A su vez, para asegurar un correcto funcionamiento será vital que los directorios recién creados posean todos los permisos existentes.

Tal y como se ha mencionado anteriormente, la configuración de Apache Cassandra se encuentra descrita en el fichero cassandra.yaml. Los valores que poseen ciertos atributos de este fichero han de ser reasignados en cada uno de los nodos para que Cassandra opere correctamente. Antes de mostrar la tabla XXX que agrupa dichos valores en cada nodo, se procederá a definirlos con la intención de entender la importancia que tienen en Cassandra.

El nombre del cluster; usado para que las máquinas de un cluster lógico no se mezclen con otro. Todos los nodos del cluster deben de tener el mismo valor.

Se utiliza cuando el nodo tiene un rango contiguo en el anillo. Existen herramientas que especificando el número de nodos que componen el cluster calculan el valor del token para cada uno de estos.

Una lista de direcciones IP delimitados por coma que se utilizan como punto de contacto para cuando un nodo se une al cluster. Cassandra también utiliza esta lista para aprender la topología del anillo.

La dirección IP que utilizan otros nodos para conectarse a una máquina especifica. Si no se indica nada, el nombre de la máquina tiene que conducir a su IP utilizando el fichero etc/hostname.

La dirección IP de escucha para conexiones de cliente. El valor por defecto es localhost y sus valores posibles son:

- 0.0.0.0: Escucha en todas la interfaces configuradas

- Dirección IP
- Nombre de máquina
- Sin especificar: el nombre de la máquina tiene que conducir a su IP utilizando el fichero etc/hostname

Establece el modo en el que Cassandra localiza nodos y envía peticiones de enrutamiento. Los más utilizados son los siguientes y a la hora de configurar los nodos, todos han de tener un mismo valor:

- SimpleSnitch: Se utiliza solo para la implementación de centro de datos únicos.
- RackInferredSnitch: Determina la ubicación de los nodos por rack o por data center.

Dichos atributos han sido configurados de la siguiente manera en el fichero cassandra.yaml de cada nodo del cluster:

Cassandra ofrece herramientas como nodetool para comprobar que este ha sido instalado de forma correcta a través del cluster.

// mencionar como comprobar mediante nodetool que todo ha ido bien

3.4.2. Configuración de Apache Spark

///Cambiar los hostname de las máquinas

Asignar nombre univoco a cada máquina es otro de los pasos que ha de realizarse. Ello se logra modificando el fichero /etc/hostname y cambiando el valor que tiene por defecto (ubuntu) por el nombre con el que se quiera bautizar.

La configuración de Apache Spark puede ser realizada tanto de forma manual como automática, pero por la sencillez y comodidad de ofrece esta vez se ha elegido configurarlo de la segunda manera.

Apache Spark ofrece una serie de scripts como start-all y stop-all que ejecutados en el nodo maestro permiten poner en marcha o parar Spark en todos los nodos del cluster.

Para que estos scripts funcionen de forma correcta será necesario hacer una pequeña configuración. Dentro de la carpeta conf de todos los nodos de la infraestructura se ha de crear un fichero llamado slaves, y dentro, especificar el hostname de las maquinas que vayan a trabajar como tal.

Bibliografía y Referencias

[1] Tushar Deepak Chandra y Sam Toueg. “Unreliable failure detectors for reliable distributed systems”. En: *Journal of the ACM (JACM)* 43.2 (1996), págs. 225-267.

[2] Fay Chang y col. “Bigtable: A distributed storage system for structured data”. En: *ACM Transactions on Computer Systems (TOCS)* 26.2 (2008), pág. 4.

[3] Jeffrey Dean y Sanjay Ghemawat. “MapReduce: simplified data processing on large clusters”. En: *Communications of the ACM* 51.1 (2008), págs. 107-113.

[4] Giuseppe DeCandia y col. “Dynamo: amazon’s highly available key-value store”. En: *ACM SIGOPS Operating Systems Review* 41.6 (2007), págs. 205-220.

[5] Alan Demers y col. “Epidemic algorithms for replicated database maintenance”. En: *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*. ACM. 1987, págs. 1-12.

[6] Seth Gilbert y Nancy Lynch. “Brewer’s conjecture and the feasibility of consistent, available, partition-tolerant web services”. En: *ACM SIGACT News* 33.2 (2002), págs. 51-59.

[7] Marcel JE Golay. “Notes on digital coding”. En: *Proceedings of the Institute of Radio Engineers* 37.6 (1949), págs. 657-657.

[8] Naohiro Hayashibara y col. “The/spl phi/accrual failure detector”. En: *Reliable Distributed Systems, 2004. Proceedings of the 23rd IEEE International Symposium on*. IEEE. 2004, págs. 66-78.

[9] Avinash Lakshman y Prashant Malik. “Cassandra: a decentralized structured storage system”. En: *ACM SIGOPS Operating Systems Review* 44.2 (2010), págs. 35-40.

[10] James Manyika y col. “Big data: The next frontier for innovation, competition, and productivity”. En: (2011).

[11] Ralph C Merkle. “A digital signature based on a conventional encryption function”. En: *Conference on the Theory and Application of Cryptographic Techniques*. Springer. 1987, págs. 369-378.

[12] Raghunath Nambiar. “Towards an Industry Standard for Benchmarking Big Data Workloads”. En: ().

[13] Bruce Jay Nelson. *Remote procedure call*. Inf. téc. Carnegie-Mellon Univ. Dept. Comput. Sci., 1981.

[14] Tilmann Rabl y col. “Solving big data challenges for enterprise application performance management”. En: *Proceedings of the VLDB Endowment* 5.12 (2012), págs. 1724-1735.

[15] Alexandru Dan SICOE y col. *A Persistent Back-End for the ATLAS Online Information Service (P-BEAST)*. Inf. téc. ATL-COM-DAQ-2012-010, 2012.

[16] Mark Slee, Aditya Agarwal y Marc Kwiatkowski. “Thrift: Scalable cross-language services implementation”. En: *Facebook White Paper* 5.8 (2007).

[17] *The State of Big Data in the Large Corporate World*. Big Data Executive Survey 2013 Boston. 2013.

[18] Reynold S Xin y col. “Shark: SQL and rich analytics at scale”. En: *Proceedings of the 2013 ACM SIGMOD International Conference on Management of data*. ACM. 2013, págs. 13-24.

[19] Matei Zaharia y col. “Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing”. En: *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association. 2012, págs. 2-2.

[20] Matei Zaharia y col. “Spark: cluster computing with working sets.” En: *HotCloud* 10 (2010), págs. 10-10.